Vision-IA

Utilizar IA e uma camera de segurança para monitorar uma loja.

Artigo feito e desenvolvido por Marlon Sousa
https://marlonsousa.medium.com
https://marlonsousa.science.blog

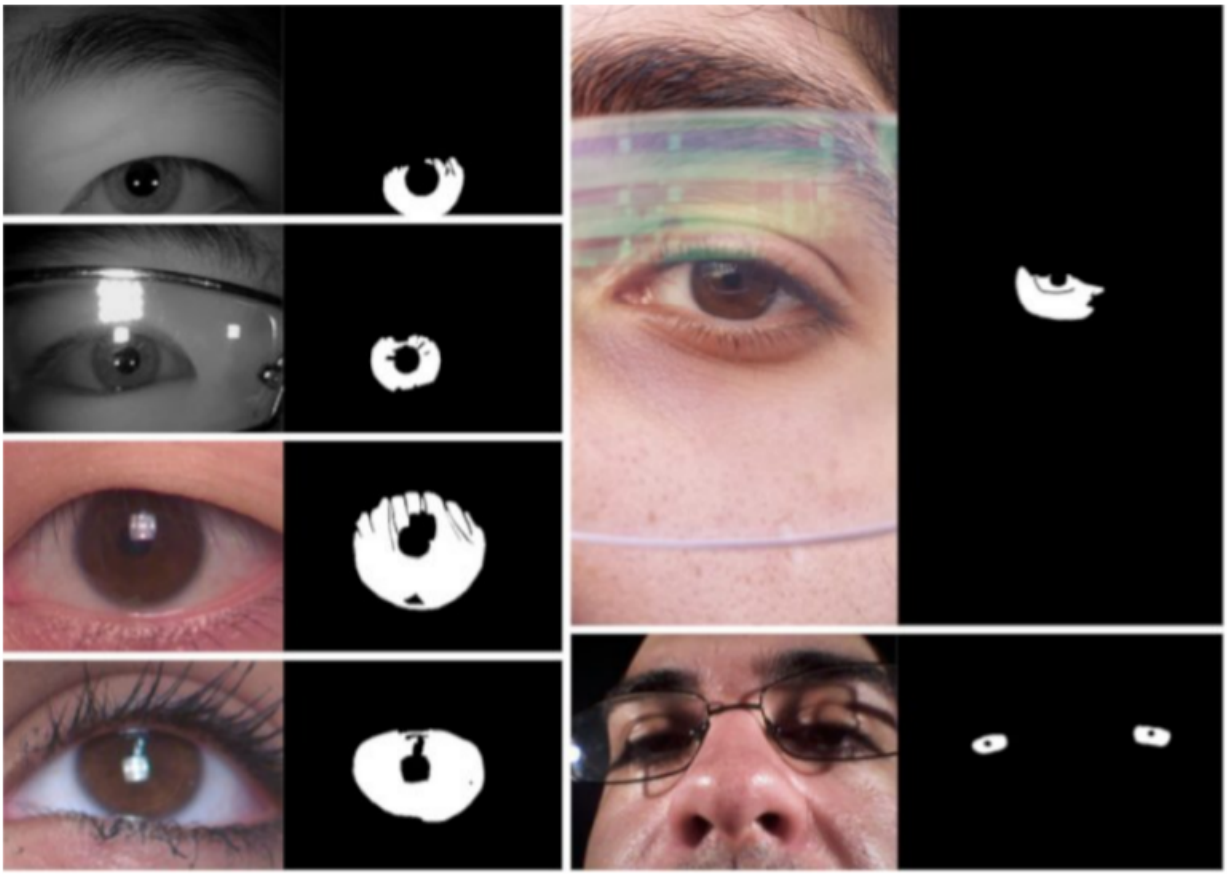
Convolutional Neural Network

Utilizaremos um dateset que criei, e nossa inteligência será capaz de detectar assaltos em lojas.

Para esta pesquisa seria ideal fazer simulações de assalto e compra normal dentro da loja em que nossa aplicação será usada.

A rede neural convolucional utiliza e processa imagens para vetores para classificar as images.

Aqui por exemplo, a inteligência consegue detectar um carro e diferenciar de um caminhão.



A IA na biometria

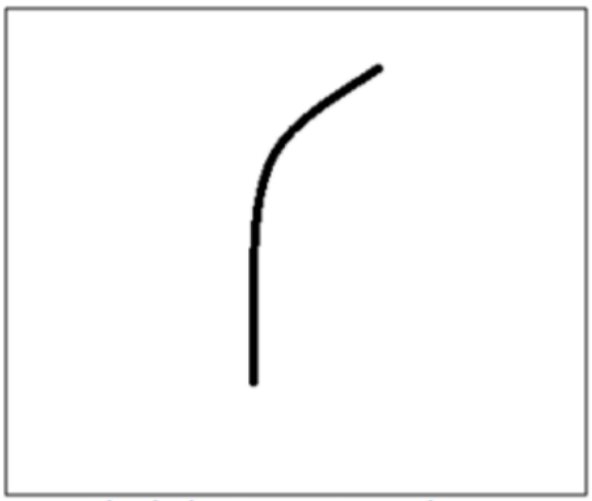
A CNN is topically composed by four types of layers: Convolutional Pooling Relu Fully Connected

A CNN é composta por algumas camadas em sua rede neural

- Convolucional
- Pooling
- Flatten

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

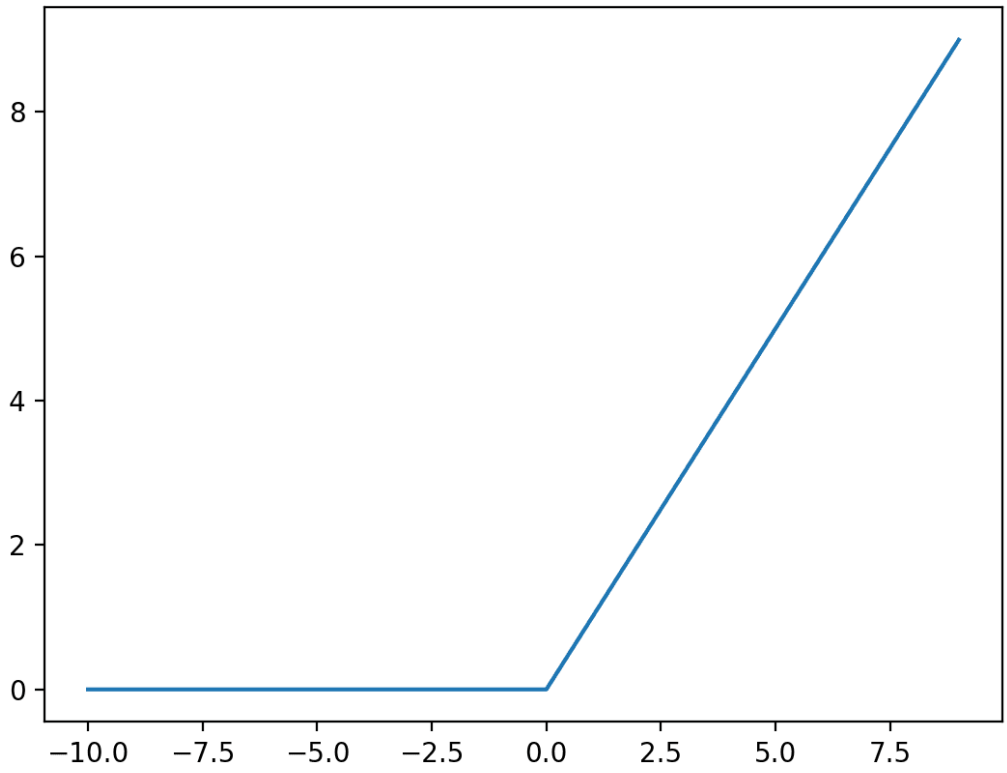
Pixel representation of filter



Visualization of a curve detector filter

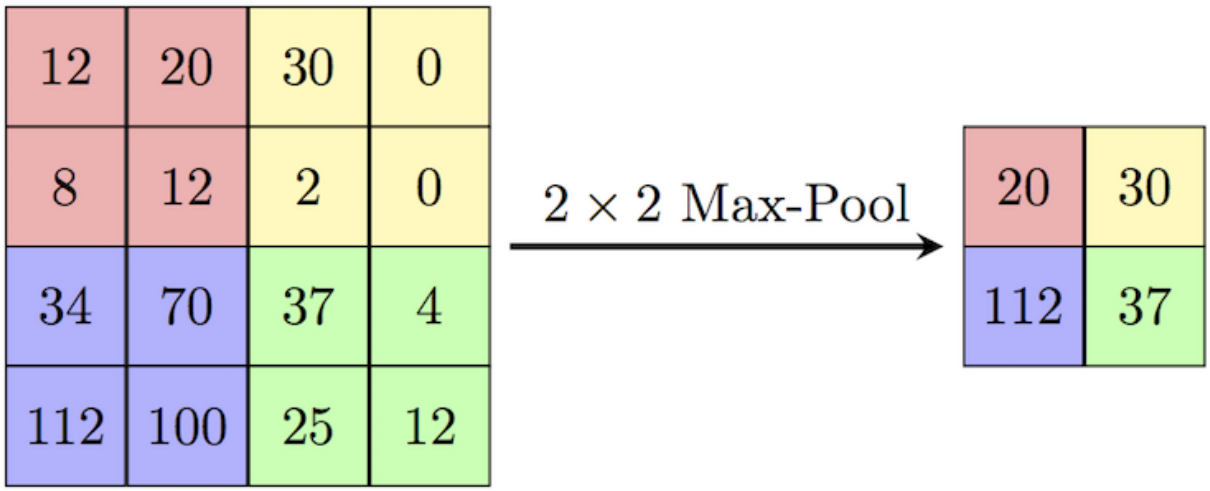
A camada convolucional pega as camadas de RGB e converte em uma matriz de pixel, como vemos na imagem.

Esses valores mudam quando utilizamos camadas de filtros, mas não veremos isso agora.

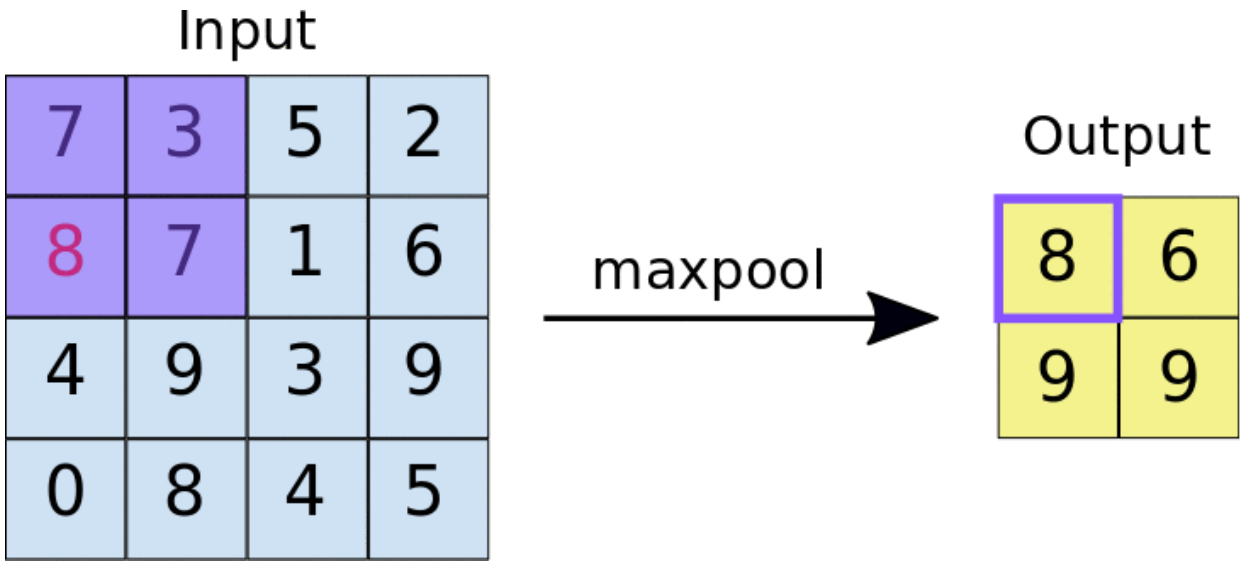


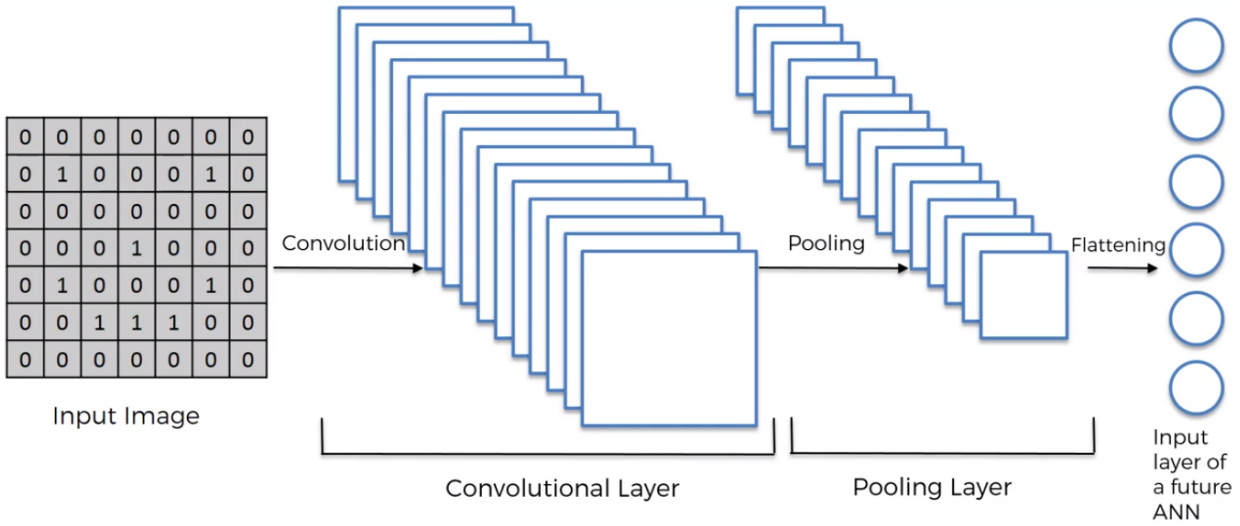
A função Relu é utilizada nas primeiras camadas da rede neural.

$0, para\ x < 0$
$x, para\ x \geq 1$



O maxpooling (2D) fá uma redução na dimensionalidade da imagem, isto reduzirá o processamento.

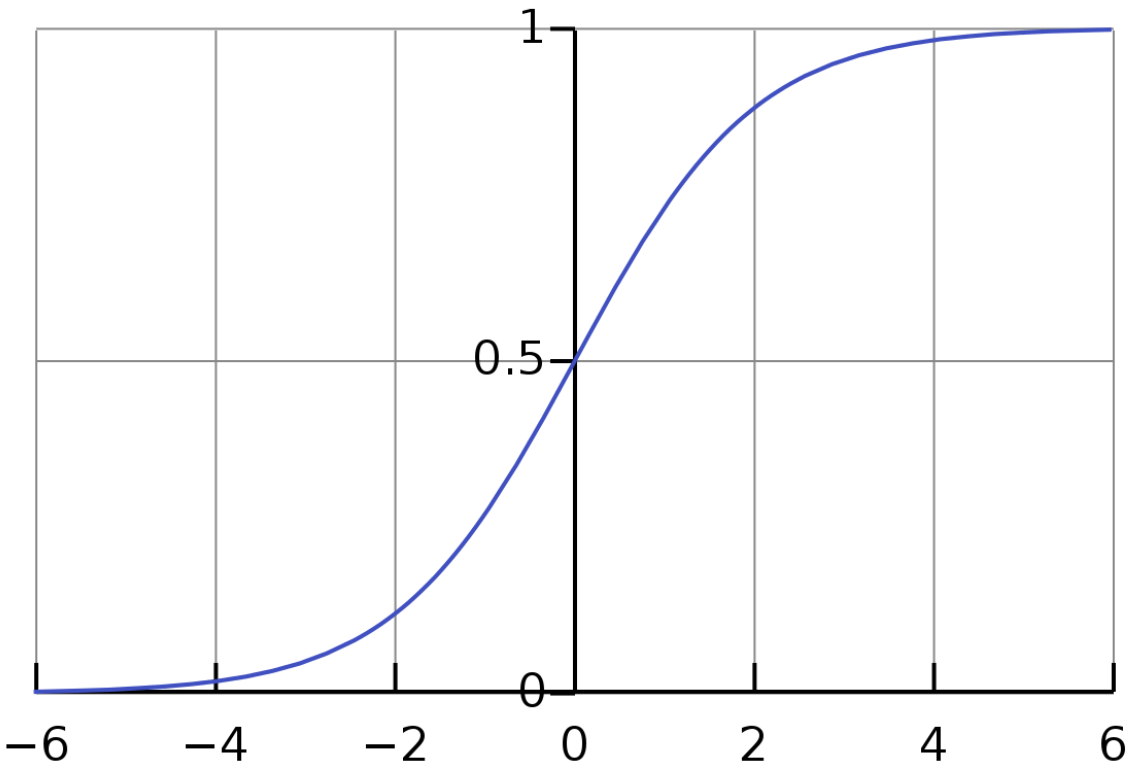




O Flatten fará a conversão para um array, e será nossa última camada de CNN. Após isto faremos a aplicação da Rede Neural Densa.

Na Prática

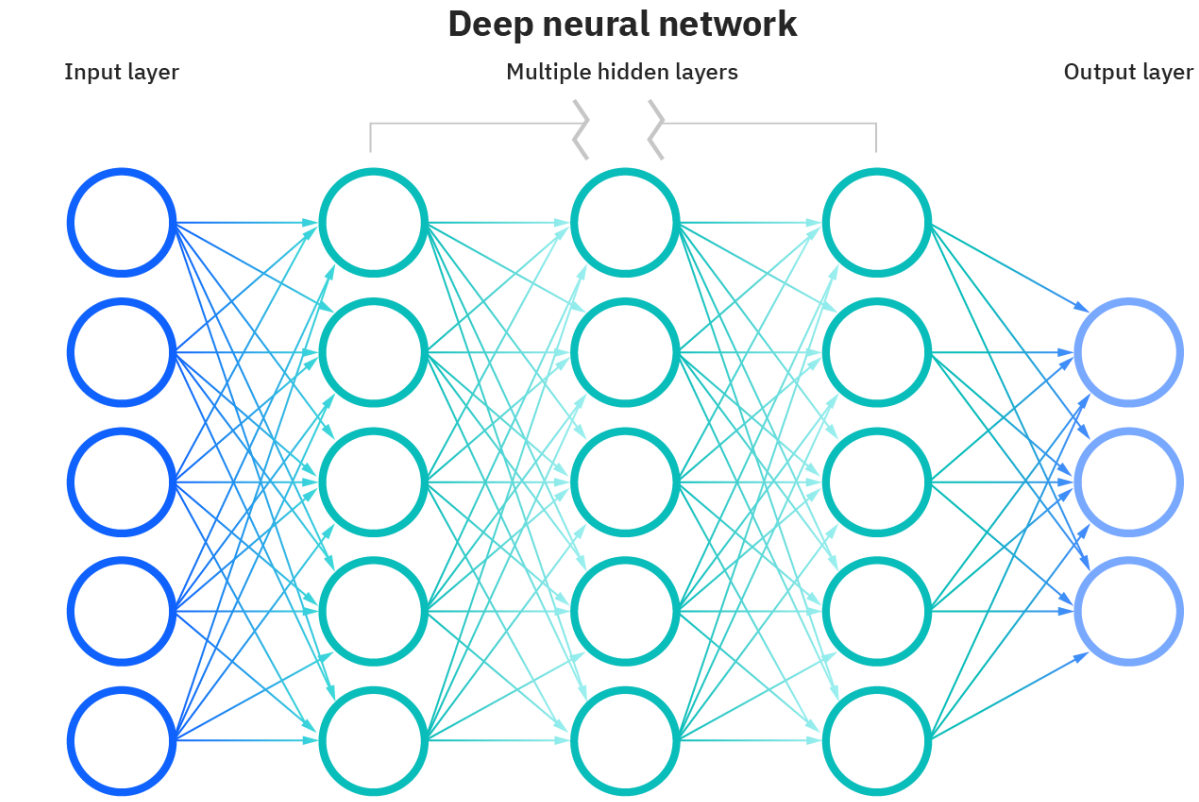
Nossa inteligência será capaz de detectar possíveis assaltos nas lojas e avisar o dono ou às altoridades.



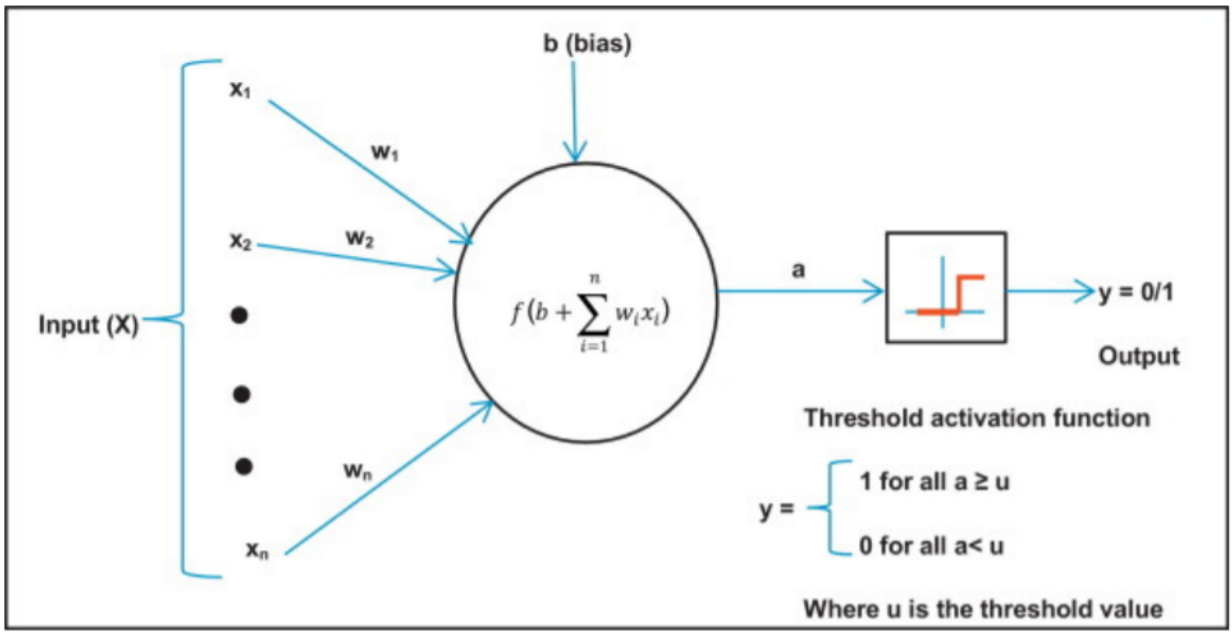
Como estamos tratando de duas classes (assalto, não assalto), nós usaremos a função sigmoid.

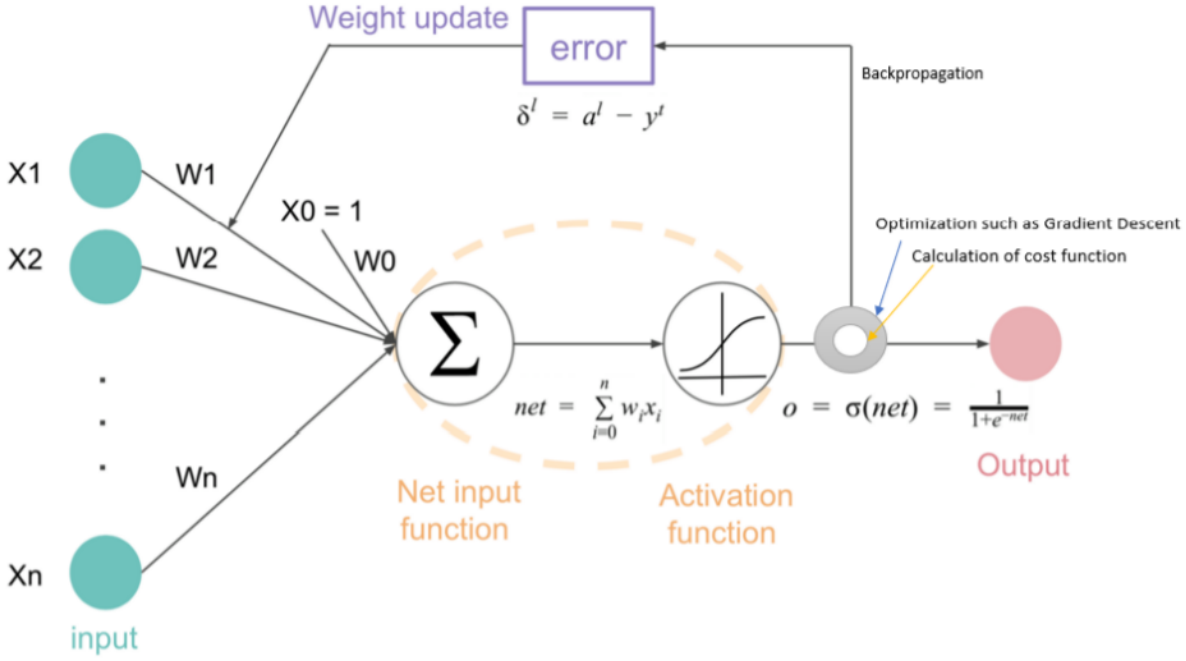
$$\frac{1}{1+e^{-x}}$$

Rede Neural Densa



- Teremos a camada de entrada
- As camadas ocultas
- As camadas de saída, que terá duas possibilidades





Nossa rede neural fará o treino dos pesos para a classificação.

PS. Este artigo visa mostrar apenas como esta IA virá a funcionar, não mostraremos detalhes matemáticos sobre o funcionamento, e não mostraremos todas as funções de ativação (softmax, linear, etc.).

Classificação

- Assalto
- Normal

Programação

Este será um modelo fraco pois é apenas um exemplo desenvolvido por mim

Importações

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from keras.preprocessing import image
```

Modelo da Rede Neural Convolucional

```
classificador = Sequential()
classificador.add(Conv2D(32, (3,3), input_shape = (64, 64, 3), activation =
classificador.add(BatchNormalization())
classificador.add(MaxPooling2D(pool_size = (2,2)))
```

Aqui temos a camada de convolução (2D), utilizando 32 filtros diferentes.
O BatchNormalization é uma técnica muito utilizada para padroniza as entradas em uma camada para cada minilote.
O MaxPooling2D faz a redução de dimensionalidade como falamos antes.

```
classificador.add(Conv2D(32, (3,3), input_shape = (64, 64, 3), activation =
classificador.add(BatchNormalization())
classificador.add(MaxPooling2D(pool_size = (2,2)))
classificador.add(Flatten())
```

Aqui faremos mais uma camada de Convolução, veja que as ativações que estamos utilizando e a relu.
O Flatten é adicionado apenas no final de nossa camada de convolução e pooling, pois ela fara a vetorização.

```
classificador.add(Dense(units = 128, activation = 'relu'))
classificador.add(Dropout(0.2))
classificador.add(Dense(units = 128, activation = 'relu'))
classificador.add(Dropout(0.2))
classificador.add(Dense(units = 1, activation = 'sigmoid'))
```

Por último faremos nossa camada densa. Utilizaremos o Dropout para evitar o overfitting no nosso modelo, ele fará o drop de algumas camadas aleatórias no nosso modelo.

Na nossa última linha temos a camada de saída que terá apenas uma saída. A ativação utilizada é a sigmoid, pois como falei estamos utilizando apenas duas classes.

```
classificador.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                      metrics = ['accuracy'])
```

Aqui faremos o compile do nosso modelo.

- Optimizer - Adam
- loss - bynary_crossentropy, pois estamos trantando de duas classes
- metrics - accuracy

```
train_gen = ImageDataGenerator(rescale = 1./255,
                               rotation_range = 7,
                               horizontal_flip = True,
                               shear_range = 0.2,
                               height_shift_range = 0.07,
                               zoom_range = 0.2)
test_gen = ImageDataGenerator(rescale = 1./255)
```

Aqui faremos o gerador de treino

```
base_treinamento = train_gen.flow_from_directory('dataset/training_set',
                                                 target_size = (6
                                                 batch_size = 32,
                                                 class_mode = 'bi
base_teste = gerador_teste.flow_from_directory('dataset/test_set',
                                               target_size = (64, 64),
                                               batch_size = 32,
                                               class_mode = 'binary')
```

Aqui buscaremos nosso diretório de treino e teste, onde se encontram nossas imagens de “assalto” e “normal”.

```
classificador.fit_generator(base_treinamento, steps_per_epoch = 4000 / 32,
                            epochs = 10, validation_data = base_teste,
                            validation_steps = 1000 / 32)
```

Aqui faremos por fim o treino, que rodará 3 vezes.

Quanto maior o epochs, melhor o resultado, mas você precisará de uma máquina muito potente para isso.

```
imagem_teste = image.load_img('dataset/test_set/assalto/assalto1.jpg',
                              target_size = (64,64))
```

Aqui faremos nosso teste na mão, passamos o caminho da nossa foto que queremos testar.

```
imagem_teste = image.img_to_array(imagem_teste)
imagem_teste /= 255
imagem_teste = np.expand_dims(imagem_teste, axis = 0)
previsao = classificador.predict(imagem_teste)
```

Vale lembra que quando mais próximo de 1, maior a chance de ser um assalto.

Agora o que você terá que fazer e fazer plug dessa aplicação em uma outra aplicação de CV2 para a camera de segurança.

CV2 Python

OpenCV (Open Source Computer Vision Library) é uma bibliotéca do python para aplicações com visão computacional.

```
$ pip install opencv-python
```

Fazemos a instalação do opencv

```
import cv2
```

Faremos a importação

```
import cv2
vid = cv2.VideoCapture(0)
while(True):
    ret, frame = vid.read()
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord ( 'q' ):
        break
vid.release()
cv2.destroyAllWindows()
```

Captura da Camera

Agora é só fazer o plug entre nossas aplicações